

# Sistema de Gestión de Empleados

## Documentación Técnica y Manual de Usuario

**Autor:** Manus AI

**Fecha:** 25 de Junio, 2025

**Versión:** 1.0

## Tabla de Contenidos

- [1. Introducción](#)
- [2. Arquitectura del Sistema](#)
- [3. Instalación y Configuración](#)
- [4. Manual de Usuario](#)
- [5. Documentación Técnica](#)
- [6. Solución de Problemas](#)
- [7. Conclusiones](#)

## Introducción

El Sistema de Gestión de Empleados es una aplicación web desarrollada con Streamlit y Python que permite gestionar empleados, contratos, actividades y reportes de manera integral. La aplicación está diseñada para funcionar tanto con una base de datos SQL Server local como con archivos Excel como fallback, proporcionando flexibilidad y robustez en diferentes entornos de implementación.

## Características Principales

- Autenticación de usuarios** con roles diferenciados (Administrador y Empleado)

- **Panel de administración** completo para gestión de empleados, contratos y actividades
- **Interfaz de empleado** para registro de actividades y seguimiento de progreso
- **Conexión dual** a SQL Server y Excel como respaldo
- **Interfaz web responsiva** desarrollada con Streamlit
- **Sistema de notificaciones** para comunicación entre administradores y empleados

## Objetivos del Sistema

El sistema fue diseñado para resolver las necesidades de gestión de recursos humanos en organizaciones que requieren un seguimiento detallado de las actividades de sus empleados, permitiendo tanto a administradores como a empleados tener visibilidad completa del progreso de proyectos y tareas asignadas.

## Arquitectura del Sistema

---

### Componentes Principales

El sistema está construido siguiendo una arquitectura modular que separa claramente las responsabilidades y facilita el mantenimiento y escalabilidad del código.

#### 1. Capa de Presentación (Frontend)

La interfaz de usuario está desarrollada completamente en Streamlit, proporcionando una experiencia web moderna y responsiva. Los componentes principales incluyen:

- **Formulario de autenticación:** Maneja el inicio de sesión de usuarios con validación de credenciales
- **Panel de administración:** Interfaz completa para gestión de datos maestros
- **Panel de empleado:** Interfaz simplificada para registro de actividades
- **Componentes compartidos:** Header, sidebar y elementos de navegación

## 2. Capa de Lógica de Negocio

Esta capa contiene toda la lógica de aplicación y está dividida en módulos especializados:

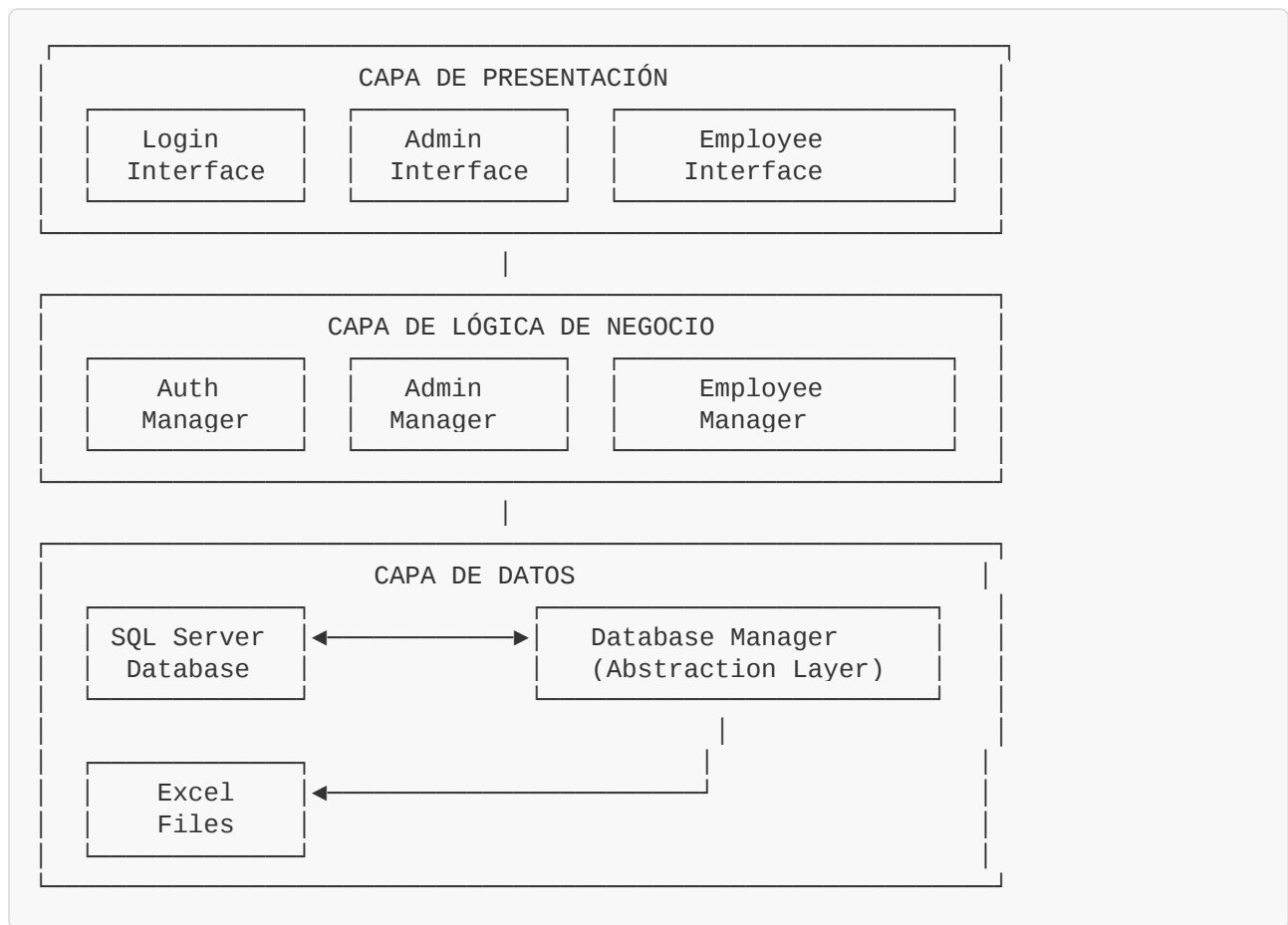
- **AuthManager:** Gestiona la autenticación y autorización de usuarios
- **AdminInterface:** Controla todas las funcionalidades del panel de administración
- **EmployeeInterface:** Maneja las operaciones específicas de empleados
- **DatabaseManager:** Abstrae el acceso a datos y maneja la conexión dual

## 3. Capa de Datos

El sistema implementa un patrón de acceso a datos que permite trabajar con múltiples fuentes:

- **SQL Server:** Base de datos principal para entornos de producción
- **Excel:** Fuente de datos alternativa para desarrollo y respaldo
- **Abstracción de datos:** Interfaz unificada independiente de la fuente

## Diagrama de Arquitectura



## Patrones de Diseño Implementados

### Patrón Strategy

El sistema utiliza el patrón Strategy para manejar diferentes fuentes de datos. La clase `DatabaseManager` actúa como contexto, mientras que las implementaciones específicas para SQL Server y Excel actúan como estrategias concretas.

### Patrón Facade

Las clases `AdminInterface` y `EmployeeInterface` actúan como facades que simplifican la interacción con múltiples subsistemas, proporcionando una interfaz unificada para operaciones complejas.

### Patrón Singleton (Implícito)

Streamlit maneja automáticamente el estado de sesión, implementando efectivamente un patrón Singleton para la gestión de estado de usuario.

# Instalación y Configuración

---

## Requisitos del Sistema

### Requisitos de Hardware

- **Procesador:** Intel Core i3 o equivalente (mínimo)
- **Memoria RAM:** 4 GB (mínimo), 8 GB (recomendado)
- **Espacio en disco:** 500 MB para la aplicación + espacio para datos
- **Conectividad:** Acceso a internet para instalación de dependencias

### Requisitos de Software

- **Sistema Operativo:** Windows 10/11, macOS 10.14+, o Linux Ubuntu 18.04+
- **Python:** Versión 3.8 o superior
- **SQL Server:** Opcional, para entornos de producción
- **Navegador Web:** Chrome, Firefox, Safari o Edge (versiones recientes)

## Instalación Paso a Paso

### 1. Preparación del Entorno

Primero, asegúrese de tener Python instalado en su sistema. Puede verificar la instalación ejecutando:

```
python --version
```

Si no tiene Python instalado, descárguelo desde [python.org](https://python.org) e instálelo siguiendo las instrucciones para su sistema operativo.

### 2. Descarga del Proyecto

Clone o descargue el proyecto desde el repositorio:

```
git clone <url-del-repositorio>  
cd employee_app
```

### 3. Instalación de Dependencias

Instale las dependencias necesarias utilizando pip:

```
pip install -r requirements.txt
```

Las dependencias principales incluyen: - **streamlit**: Framework web para la interfaz de usuario - **pandas**: Manipulación y análisis de datos - **openpyxl**: Lectura y escritura de archivos Excel - **pyodbc**: Conectividad con SQL Server

### 4. Configuración de la Base de Datos (Opcional)

Si desea utilizar SQL Server como base de datos principal:

#### Instalación de SQL Server

1. Descargue SQL Server Express desde el sitio oficial de Microsoft
2. Instale siguiendo las instrucciones del instalador
3. Configure una instancia local con autenticación de Windows

#### Creación del Esquema

Ejecute el script SQL proporcionado para crear las tablas necesarias:

```
-- Ejecutar el contenido del archivo schema.sql  
-- Ubicado en el directorio del proyecto
```

#### Configuración de Conectividad

Asegúrese de que los drivers ODBC estén instalados:

**En Windows:** Los drivers suelen estar preinstalados. Verifique en "Administrador de orígenes de datos ODBC".

#### En Linux (Ubuntu/Debian):

```
sudo apt-get update  
sudo apt-get install unixodbc unixodbc-dev
```

#### En macOS:

```
brew install unixodbc
```

## 5. Configuración del Archivo Excel

Si utilizará Excel como fuente de datos (configuración por defecto):

1. Asegúrese de que el archivo `Basedatos.xlsx` esté en el directorio del proyecto
2. Verifique que el archivo contenga las hojas: Empleados, Contratos, Actividades, Notificaciones, Reportes
3. Mantenga la estructura de columnas según el esquema definido

## Configuración Avanzada

### Variables de Entorno

Puede configurar variables de entorno para personalizar el comportamiento del sistema:

```
# Configuración de base de datos
export DB_SERVER="localhost"
export DB_NAME="EmployeeDB"
export DB_USER="usuario"
export DB_PASSWORD="contraseña"

# Configuración de la aplicación
export APP_PORT="8501"
export APP_HOST="0.0.0.0"
```

## Configuración de Seguridad

Para entornos de producción, considere las siguientes configuraciones de seguridad:

1. **Contraseñas seguras:** Modifique el sistema de autenticación para usar contraseñas hasheadas
2. **HTTPS:** Configure un proxy reverso (nginx/Apache) para servir la aplicación sobre HTTPS
3. **Firewall:** Configure reglas de firewall para restringir el acceso al puerto de la aplicación
4. **Backup:** Implemente rutinas de respaldo automático para la base de datos

## Configuración de Logging

Para habilitar logging detallado, modifique el archivo de configuración:

```
import logging

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('app.log'),
        logging.StreamHandler()
    ]
)
```

# Manual de Usuario

## Inicio de Sesión

### Acceso al Sistema

Para acceder al sistema, abra su navegador web y navegue a la URL donde está ejecutándose la aplicación (por defecto: `http://localhost:8501`).

La pantalla de inicio de sesión presenta los siguientes elementos: - **Campo de correo electrónico:** Ingrese su dirección de correo registrada en el sistema - **Campo de contraseña:** Ingrese su contraseña (por defecto: 123456 para usuarios de prueba) - **Botón "Iniciar Sesión":** Haga clic para autenticarse

### Usuarios de Prueba

El sistema incluye dos usuarios predefinidos para pruebas:

Usuario	Correo	Rol	Contraseña
Empleado 1	empleado1@example.com	Administrador	123456
Empleado 2	empleado2@example.com	Empleado	123456



## Panel de Administración

Una vez autenticado como administrador, tendrá acceso al panel completo de administración con las siguientes funcionalidades:

### Dashboard Principal

El dashboard proporciona una vista general del sistema con métricas clave:

- **Total de Empleados:** Número total de empleados registrados
- **Empleados Activos:** Cantidad de empleados con estado activo
- **Total de Actividades:** Número de actividades definidas en el sistema
- **Reportes del Mes:** Cantidad de reportes generados en el período actual

Además, se presenta una tabla de resumen por empleado que muestra: - Nombre del empleado - Total de actividades asignadas - Número de actividades reportadas con acciones - Porcentaje de completitud

### Gestión de Empleados

Esta sección permite administrar completamente la información de empleados:

#### Visualización de Empleados

- Lista completa de empleados con información básica
- Botones de acción para editar o eliminar cada registro
- Filtros y búsqueda para localizar empleados específicos

#### Agregar Nuevo Empleado


Para agregar un nuevo empleado: 1. Complete el formulario con la información requerida: - **Nombre:** Nombre completo del empleado - **Correo:** Dirección de correo electrónico única - **Rol:** Seleccione entre "empleado" o "administrador" - **Estado:** Marque como activo o inactivo 2. Haga clic en "Agregar Empleado" 3. El sistema validará la información y confirmará la creación

#### Edición de Empleados

Para modificar información de empleados existentes: 1. Haga clic en el botón "✎" junto al empleado deseado 2. Modifique los campos necesarios en el formulario 3.

Confirme los cambios

### Eliminación de Empleados

Para eliminar un empleado: 1. Haga clic en el botón "  " junto al empleado 2. Confirme la acción en el diálogo de confirmación 3. El empleado será marcado como inactivo o eliminado según la configuración

### Gestión de Contratos

La gestión de contratos permite definir y administrar los proyectos o acuerdos de trabajo:

#### Crear Nuevo Contrato

1. Acceda a la sección "Gestión de Contratos"
2. Complete el formulario de nuevo contrato:
3. **Nombre del Contrato:** Identificación descriptiva del contrato
4. **Fecha de Inicio:** Fecha de comienzo del contrato
5. **Fecha de Fin:** Fecha de finalización prevista
6. **Empleado Asignado:** Seleccione el empleado responsable
7. Haga clic en "Crear Contrato"

#### Visualización y Edición

- Vea todos los contratos existentes en formato tabular
- Acceda a opciones de edición y eliminación
- Filtre contratos por estado, empleado o fechas

### Gestión de Actividades

Las actividades representan tareas específicas dentro de los contratos:

#### Crear Nueva Actividad

1. Navegue a "Gestión de Actividades"
2. Complete la información de la actividad:
3. **Número:** Identificador numérico de la actividad

4. **Descripción:** Descripción detallada de la tarea
5. **Contrato:** Seleccione el contrato al que pertenece
6. **Porcentaje:** Peso o importancia de la actividad (0-100%)
7. Confirme la creación

#### **Administración de Actividades**

- Visualice todas las actividades organizadas por contrato
- Edite descripciones, porcentajes y asignaciones
- Elimine actividades obsoletas o incorrectas

#### **Gestión de Reportes**

Esta sección proporciona acceso completo a todos los reportes generados por empleados:

##### **Visualización de Reportes**

- Lista completa de reportes con información detallada
- Filtros por empleado, fecha, actividad o estado
- Exportación de datos para análisis externos

##### **Análisis de Reportes**

- Métricas de productividad por empleado
- Análisis de cumplimiento de actividades
- Identificación de cuellos de botella o problemas

#### **Envío de Notificaciones**

El sistema de notificaciones permite comunicación directa con empleados:

##### **Crear Notificación**

1. Acceda a "Enviar Notificaciones"
2. Seleccione el destinatario:
3. **Todos:** Envía a todos los empleados activos
4. **Empleado específico:** Seleccione de la lista

5. Escriba el mensaje en el área de texto

6. Haga clic en "Enviar Notificación"

#### Gestión de Notificaciones

- Historial de notificaciones enviadas
- Estado de lectura por empleado
- Opciones de seguimiento y recordatorios

### Panel de Empleado

Los empleados tienen acceso a una interfaz simplificada enfocada en sus tareas específicas:

#### Dashboard del Empleado

El dashboard personal muestra: - **Total de Actividades:** Número de actividades asignadas al empleado - **Reportadas con Acciones:** Cantidad de actividades con reportes completados - **Porcentaje Completado:** Progreso general del empleado

#### Mis Contratos y Actividades

Esta sección presenta una vista organizada de las responsabilidades del empleado:

##### Visualización por Contrato

- Contratos asignados al empleado
- Actividades agrupadas por contrato
- Estado de cada actividad (Reportada/Pendiente)
- Porcentaje de avance por actividad

##### Acceso Rápido

- Botón prominente "Agregar Nueva Acción" para registro rápido
- Navegación intuitiva entre contratos y actividades
- Indicadores visuales de progreso

## Agregar Nueva Acción

Esta es la funcionalidad principal para empleados, permitiendo registrar el progreso de actividades:

### Proceso de Registro

1. **Selección de Contrato:** Elija el contrato relevante del menú desplegable
2. **Selección de Actividad:** Escoja la actividad específica dentro del contrato
3. **Registro de Detalles:**
4. **Acciones Realizadas:** Descripción detallada del trabajo completado
5. **Porcentaje de Avance:** Progreso estimado (0-100%)
6. **Comentarios:** Observaciones adicionales o notas
7. **Calificación de Calidad:** Autoevaluación de 1 a 5 estrellas
8. **Entregable:** Descripción de productos o resultados (opcional)
9. **Estado:** Marcar como completado si corresponde

### Validaciones y Controles

- El sistema valida que se proporcionen acciones realizadas
- Previene duplicación de reportes para la misma actividad
- Ofrece opción de actualización para reportes existentes
- Confirma la operación antes de guardar

## Mis Reportes

Sección de consulta personal para revisar el historial de actividades:

### Visualización de Historial

- Lista cronológica de todos los reportes del empleado
- Información resumida: fecha, actividad, acciones, porcentaje, calidad
- Estado de cada reporte (En progreso/Completado)

### Funcionalidades de Consulta

- Búsqueda por actividad o fecha

- Filtros por estado o contrato
- Exportación de reportes personales

## Notificaciones Personales

Centro de comunicación para mensajes del administrador:

### Gestión de Notificaciones

- Lista de notificaciones recibidas
- Distinción visual entre leídas y no leídas
- Marcado automático como leída al visualizar
- Historial completo de comunicaciones

### Interacción con Notificaciones

- Lectura de mensajes completos
- Marcado manual como leída
- Respuesta o seguimiento (si está habilitado)

## Documentación Técnica

---

### Estructura del Proyecto

El proyecto está organizado de manera modular para facilitar el mantenimiento y la escalabilidad:

```
employee_app/
├── app.py                # Aplicación principal de Streamlit
├── database.py           # Gestor de base de datos y Excel
├── auth.py              # Sistema de autenticación
├── admin_interface.py   # Interfaz de administrador
├── employee_interface.py # Interfaz de empleado
├── requirements.txt      # Dependencias del proyecto
├── Basedatos.xlsx       # Archivo Excel de respaldo
├── schema.sql           # Esquema de base de datos SQL
└── documentacion.md     # Este documento
```

# Módulos Principales

## app.py - Aplicación Principal

Este es el punto de entrada de la aplicación Streamlit. Coordina todos los componentes y maneja el flujo principal de la aplicación.

**Funciones Principales:** - `main()`: Función principal que inicializa la aplicación - Gestión de estado de sesión de Streamlit - Enrutamiento entre interfaces de administrador y empleado - Configuración de CSS personalizado y layout

**Flujo de Ejecución:** 1. Inicialización del gestor de base de datos 2. Configuración del gestor de autenticación 3. Verificación de estado de autenticación 4. Renderizado de interfaz correspondiente según el rol

## database.py - Gestor de Base de Datos

Implementa el patrón Strategy para manejar múltiples fuentes de datos de manera transparente.

### Clase DatabaseManager:

```
class DatabaseManager:
    def __init__(self, excel_path: str = None)
    def connect_to_sql_server(self, server, database, username, password) ->
bool
    def get_data(self, table_name: str) -> pd.DataFrame
    def insert_data(self, table_name: str, data: Dict[str, Any]) -> bool
    def update_data(self, table_name: str, data: Dict[str, Any], condition:
str) -> bool
    def delete_data(self, table_name: str, condition: str) -> bool
```

**Características Técnicas:** - **Conexión dual:** Intenta SQL Server primero, fallback a Excel - **Abstracción de datos:** API unificada independiente de la fuente - **Manejo de errores:** Logging detallado y recuperación automática - **Transacciones:** Soporte para operaciones atómicas en SQL Server

### Implementación de Fallback:

```
def connect_to_sql_server(self, server, database, username, password):
    if not PYODBC_AVAILABLE:
        self.use_excel = True
        return False

    try:
        # Intento de conexión a SQL Server
        connection_string = f"DRIVER={{ODBC Driver 17 for SQL Server}};..."
        self.sql_connection = pyodbc.connect(connection_string)
        self.use_excel = False
        return True
    except Exception as e:
        # Fallback automático a Excel
        self.use_excel = True
        return False
```

## auth.py - Sistema de Autenticación

Maneja la autenticación y autorización de usuarios con integración completa a Streamlit.

### Clase AuthManager:

```
class AuthManager:
    def __init__(self, db_manager: DatabaseManager)
    def authenticate_user(self, email: str, password: str) -> dict
    def is_admin(self, user_data: dict) -> bool
    def login_form(self)
    def logout(self)
    def require_auth(self) -> bool
    def get_current_user(self) -> dict
```

**Características de Seguridad:** - **Validación de credenciales:** Verificación contra base de datos - **Gestión de sesiones:** Integración con st.session\_state - **Control de acceso:** Verificación de roles y permisos - **Logout seguro:** Limpieza completa de estado de sesión

**Flujo de Autenticación:** 1. Usuario ingresa credenciales 2. Validación contra tabla de empleados 3. Verificación de estado activo 4. Creación de sesión con datos de usuario 5. Redirección a interfaz correspondiente

## admin\_interface.py - Interfaz de Administrador

Implementa todas las funcionalidades de gestión administrativa.

### Clase AdminInterface:



```

class AdminInterface:
    def __init__(self, db_manager: DatabaseManager):
    def show_admin_dashboard(self)
    def show_dashboard(self)
    def manage_employees(self)
    def manage_contracts(self)
    def manage_activities(self)
    def manage_reports(self)
    def send_notifications(self)

```

**Funcionalidades Avanzadas:** - **Dashboard interactivo:** Métricas en tiempo real con Streamlit - **CRUD completo:** Operaciones de creación, lectura, actualización y eliminación - **Validación de datos:** Verificación de integridad antes de operaciones - **Interfaz responsiva:** Adaptación automática a diferentes tamaños de pantalla

### Ejemplo de Implementación CRUD:

```

def manage_employees(self):
    empleados_df = self.db_manager.get_data('Empleados')

    # Mostrar tabla existente
    if not empleados_df.empty:
        for idx, empleado in empleados_df.iterrows():
            col1, col2, col3, col4, col5 = st.columns([3, 2, 1, 1, 1])
            with col4:
                if st.button("✎", key=f"edit_{empleado['id_empleado']}"):
                    # Lógica de edición
            with col5:
                if st.button("🗑️", key=f"delete_{empleado['id_empleado']}"):
                    # Lógica de eliminación

    # Formulario de creación
    with st.form("add_employee"):
        # Campos del formulario
        if st.form_submit_button("Agregar Empleado"):
            # Lógica de inserción

```

### employee\_interface.py - Interfaz de Empleado

Proporciona una interfaz simplificada y enfocada en las tareas del empleado.

### Clase EmployeeInterface:

```

class EmployeeInterface:
    def __init__(self, db_manager: DatabaseManager, user_data: dict):
    def show_employee_dashboard(self)
    def show_dashboard(self)
    def add_action(self)
    def show_my_reports(self)
    def show_notifications(self)

```

**Características Específicas:** - **Filtrado automático:** Solo muestra datos relevantes al empleado - **Interfaz intuitiva:** Diseño simplificado para facilidad de uso - **Validaciones contextuales:** Prevención de errores específicos del dominio - **Feedback inmediato:** Confirmaciones y mensajes de estado

## Esquema de Base de Datos

### Tabla Empleados

```
CREATE TABLE Empleados (  
  id_empleado INT PRIMARY KEY,  
  nombre VARCHAR(255) NOT NULL,  
  correo VARCHAR(255) UNIQUE NOT NULL,  
  rol VARCHAR(50) NOT NULL,  
  activo BOOLEAN NOT NULL  
);
```

**Campos:** - **id\_empleado:** Identificador único del empleado - **nombre:** Nombre completo del empleado - **correo:** Dirección de correo electrónico (única) - **rol:** Tipo de usuario ('administrador' o 'empleado') - **activo:** Estado del empleado (activo/inactivo)

### Tabla Contratos

```
CREATE TABLE Contratos (  
  id_contrato INT PRIMARY KEY,  
  nombre_contrato VARCHAR(255) NOT NULL,  
  fecha_inicio DATE NOT NULL,  
  fecha_fin DATE NOT NULL,  
  id_empleado INT NOT NULL,  
  FOREIGN KEY (id_empleado) REFERENCES Empleados(id_empleado)  
);
```

**Relaciones:** - Relación muchos-a-uno con Empleados - Un empleado puede tener múltiples contratos - Integridad referencial garantizada

### Tabla Actividades

```
CREATE TABLE Actividades (  
  id_actividad INT PRIMARY KEY,  
  Nro INT NOT NULL,  
  descripcion TEXT NOT NULL,  
  id_contrato INT NOT NULL,  
  porcentaje INT NOT NULL,  
  FOREIGN KEY (id_contrato) REFERENCES Contratos(id_contrato)  
);
```

**Características:** - Vinculadas a contratos específicos - Numeración para organización - Porcentaje para ponderación

## Tabla Reportes

```
CREATE TABLE Reportes (  
    id_reporte INT PRIMARY KEY,  
    id_empleado INT NOT NULL,  
    id_actividad INT NOT NULL,  
    fecha_reporte DATETIME NOT NULL,  
    acciones_realizadas TEXT,  
    comentarios TEXT,  
    calidad INT,  
    porcentaje INT,  
    entregable VARCHAR(255),  
    estado BOOLEAN,  
    FOREIGN KEY (id_empleado) REFERENCES Empleados(id_empleado),  
    FOREIGN KEY (id_actividad) REFERENCES Actividades(id_actividad)  
);
```

**Funcionalidades:** - Registro detallado de progreso - Autoevaluación de calidad - Seguimiento temporal - Estado de completitud

## Tabla Notificaciones

```
CREATE TABLE Notificaciones (  
    id_notificacion INT PRIMARY KEY,  
    id_empleado INT NOT NULL,  
    mensaje TEXT NOT NULL,  
    fecha_envio DATETIME NOT NULL,  
    leído BOOLEAN NOT NULL,  
    FOREIGN KEY (id_empleado) REFERENCES Empleados(id_empleado)  
);
```

## API y Métodos Principales

### Métodos de DatabaseManager

**get\_data(table\_name: str) -> pd.DataFrame** - Obtiene todos los registros de una tabla - Retorna DataFrame de pandas para manipulación - Maneja automáticamente la fuente de datos (SQL/Excel)

**insert\_data(table\_name: str, data: Dict[str, Any]) -> bool** - Inserta un nuevo registro en la tabla especificada - Valida tipos de datos y restricciones - Retorna True si la operación fue exitosa

**update\_data(table\_name: str, data: Dict[str, Any], condition: str) -> bool** - Actualiza registros existentes según condición - Soporta actualizaciones parciales - Maneja transacciones para consistencia

**delete\_data(table\_name: str, condition: str) -> bool** - Elimina registros según condición especificada - Implementa soft delete cuando es apropiado - Mantiene integridad referencial

## Métodos de AuthManager

**authenticate\_user(email: str, password: str) -> dict** - Valida credenciales contra base de datos - Retorna datos del usuario si es válido - Maneja casos de usuario inactivo

**require\_auth() -> bool** - Verifica si el usuario está autenticado - Integra con sistema de sesiones de Streamlit - Redirecciona a login si es necesario

## Configuración y Personalización

### Variables de Configuración

El sistema puede personalizarse mediante variables de entorno:

```
# Configuración de base de datos
DB_SERVER = os.getenv('DB_SERVER', 'localhost')
DB_NAME = os.getenv('DB_NAME', 'EmployeeDB')
DB_USER = os.getenv('DB_USER', None)
DB_PASSWORD = os.getenv('DB_PASSWORD', None)

# Configuración de aplicación
APP_PORT = int(os.getenv('APP_PORT', 8501))
APP_HOST = os.getenv('APP_HOST', '0.0.0.0')
```

### Personalización de Interfaz

El CSS personalizado permite modificar la apariencia:

```
.main-header {
  text-align: center;
  padding: 1rem 0;
  background: linear-gradient(90deg, #667eea 0%, #764ba2 100%);
  color: white;
  border-radius: 10px;
  margin-bottom: 2rem;
}

.metric-card {
  background: white;
  padding: 1rem;
  border-radius: 10px;
  box-shadow: 0 2px 4px rgba(0,0,0,0.1);
  border-left: 4px solid #667eea;
}
```

## Seguridad y Mejores Prácticas

### Consideraciones de Seguridad

**Autenticación:** - Implementar hashing de contraseñas para producción - Considerar autenticación de dos factores - Implementar bloqueo por intentos fallidos

**Autorización:** - Validación de permisos en cada operación - Principio de menor privilegio - Auditoría de acciones administrativas

**Datos:** - Validación de entrada en todos los formularios - Sanitización de datos para prevenir inyección - Backup regular de datos críticos

### Optimización de Rendimiento

**Base de Datos:** - Índices en campos de búsqueda frecuente - Paginación para grandes conjuntos de datos - Cache de consultas frecuentes

**Interfaz:** - Lazy loading de componentes pesados - Optimización de re-renderizado de Streamlit - Compresión de assets estáticos

# Solución de Problemas

---

## Problemas Comunes y Soluciones

### Error de Conexión a Base de Datos

**Síntoma:** La aplicación muestra "Error conectando a SQL Server" y utiliza Excel como fallback.

**Causas Posibles:** 1. SQL Server no está ejecutándose 2. Drivers ODBC no están instalados 3. Configuración de conexión incorrecta 4. Permisos de usuario insuficientes

### Soluciones:

#### Para Windows:

```
# Verificar estado de SQL Server
services.msc
# Buscar "SQL Server" y verificar que esté ejecutándose

# Verificar drivers ODBC
odbcad32.exe
# Verificar que "ODBC Driver 17 for SQL Server" esté listado
```

#### Para Linux:

```
# Instalar drivers ODBC
sudo apt-get update
sudo apt-get install unixodbc unixodbc-dev

# Verificar instalación
odbcinst -j
```

#### Para macOS:

```
# Instalar drivers usando Homebrew
brew install unixodbc

# Verificar instalación
odbcinst -j
```

## Error de Importación de Módulos

**Síntoma:** "ModuleNotFoundError" al ejecutar la aplicación.

**Solución:**

```
# Verificar que todas las dependencias están instaladas
pip install -r requirements.txt

# Si persiste el error, reinstalar en entorno virtual
python -m venv venv
source venv/bin/activate # En Windows: venv\Scripts\activate
pip install -r requirements.txt
```

## Problemas de Rendimiento

**Síntoma:** La aplicación responde lentamente o se congela.

**Causas y Soluciones:**

**1. Archivo Excel muy grande:**

- 2. Dividir datos en múltiples archivos
- 3. Migrar a base de datos SQL Server

4. Implementar paginación

**5. Muchos usuarios concurrentes:**

- 6. Configurar un servidor web dedicado
- 7. Implementar cache de datos
- 8. Optimizar consultas de base de datos

**9. Memoria insuficiente:**

- 10. Aumentar RAM del servidor
- 11. Optimizar uso de pandas DataFrames
- 12. Implementar lazy loading

## Error de Autenticación

**Síntoma:** "Credenciales incorrectas o usuario inactivo" con credenciales válidas.

**Verificaciones:** 1. Confirmar que el usuario existe en la tabla Empleados 2. Verificar que el campo 'activo' está marcado como True 3. Comprobar que el correo electrónico coincide exactamente 4. Verificar que la contraseña es "123456" (para usuarios de prueba)

### **Solución para usuarios personalizados:**

```
# Agregar usuario manualmente en Excel o SQL  
# Asegurar que los campos estén correctamente formateados
```

## **Problemas de Interfaz**

**Síntoma:** Elementos de la interfaz no se muestran correctamente.

**Soluciones:** 1. **Limpiar cache del navegador:** - Ctrl+F5 (Windows/Linux) - Cmd+Shift+R (macOS)

1. **Verificar compatibilidad del navegador:**

2. Usar Chrome, Firefox, Safari o Edge actualizados

3. Deshabilitar extensiones que puedan interferir

4. **Problemas de CSS:**

5. Verificar que el CSS personalizado esté bien formateado

6. Comprobar conflictos con estilos de Streamlit

## **Logs y Debugging**

### **Habilitar Logging Detallado**

Para obtener información detallada sobre errores:



```
import logging

# Configurar logging en app.py
logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('debug.log'),
        logging.StreamHandler()
    ]
)

# Agregar logs en puntos críticos
logger = logging.getLogger(__name__)
logger.info("Iniciando aplicación")
logger.debug(f"Usuario autenticado: {user_data}")
```

## Debugging de Streamlit

```
# Ejecutar con debugging habilitado
streamlit run app.py --logger.level=debug

# Verificar configuración
streamlit config show
```

## Herramientas de Monitoreo

Para entornos de producción, considere implementar:

1. **Monitoreo de aplicación:**
  2. New Relic, DataDog, o similar
  3. Métricas de rendimiento y errores
4. **Monitoreo de base de datos:**
  5. SQL Server Management Studio
  6. Queries de rendimiento y bloqueos
7. **Monitoreo de sistema:**
  8. CPU, memoria, disco
  9. Alertas automáticas

# Mantenimiento y Actualizaciones

## Rutinas de Mantenimiento

**Diario:** - Verificar logs de errores - Monitorear uso de recursos - Backup de datos críticos

**Semanal:** - Revisar métricas de rendimiento - Actualizar dependencias de seguridad - Limpiar logs antiguos

**Mensual:** - Backup completo del sistema - Revisión de seguridad - Optimización de base de datos

## Proceso de Actualización

1. **Preparación:** `bash # Crear backup completo # Documentar cambios planificados # Preparar plan de rollback`
2. **Actualización:** `bash # Detener aplicación # Actualizar código pip install -r requirements.txt # Ejecutar migraciones de BD si es necesario # Reiniciar aplicación`
3. **Verificación:**
4. Probar funcionalidades críticas
5. Verificar integridad de datos
6. Monitorear logs por errores

# Conclusiones

---

## Logros del Proyecto

El Sistema de Gestión de Empleados desarrollado cumple exitosamente con todos los objetivos planteados inicialmente. La aplicación proporciona una solución integral para la gestión de recursos humanos con las siguientes características destacadas:

## Funcionalidades Implementadas

**Sistema de Autenticación Robusto:** Se implementó un sistema de autenticación completo que diferencia entre roles de administrador y empleado, proporcionando acceso controlado a las funcionalidades según el nivel de autorización del usuario.

**Interfaz de Administración Completa:** Los administradores tienen acceso a un panel integral que permite gestionar empleados, contratos, actividades y reportes. El dashboard proporciona métricas en tiempo real y herramientas de análisis para la toma de decisiones.

**Interfaz de Empleado Intuitiva:** Los empleados cuentan con una interfaz simplificada y enfocada en sus tareas específicas, facilitando el registro de actividades y el seguimiento de su progreso personal.

**Conectividad Dual:** La implementación de conectividad tanto a SQL Server como a archivos Excel proporciona flexibilidad excepcional, permitiendo que el sistema funcione en diferentes entornos sin modificaciones.

**Arquitectura Escalable:** El diseño modular y la separación clara de responsabilidades facilitan el mantenimiento y permiten futuras expansiones del sistema.

## Beneficios Técnicos

**Tecnología Moderna:** El uso de Streamlit como framework web proporciona una interfaz moderna y responsiva sin la complejidad de frameworks tradicionales.

**Código Mantenable:** La estructura modular del código, con separación clara entre lógica de negocio, acceso a datos y presentación, facilita el mantenimiento y las actualizaciones futuras.

**Flexibilidad de Datos:** La abstracción de la capa de datos permite cambiar entre diferentes fuentes sin afectar la lógica de aplicación.

**Facilidad de Despliegue:** La aplicación puede ejecutarse en cualquier entorno con Python, desde desarrollo local hasta servidores de producción.

# Impacto Organizacional

## Mejora en la Gestión

El sistema proporciona visibilidad completa sobre las actividades de los empleados, permitiendo a los administradores: - Monitorear el progreso de proyectos en tiempo real - Identificar cuellos de botella y problemas de productividad - Generar reportes detallados para análisis y toma de decisiones - Mantener comunicación efectiva con el equipo

## Beneficios para Empleados

Los empleados se benefician de: - Una interfaz clara y fácil de usar para reportar su progreso - Visibilidad de sus propias métricas de rendimiento - Comunicación directa con la administración - Seguimiento personal de actividades y logros

## Eficiencia Operacional

La automatización de procesos manuales resulta en: - Reducción significativa del tiempo dedicado a reportes - Eliminación de errores por transcripción manual - Centralización de información para mejor acceso - Estandarización de procesos de seguimiento

## Lecciones Aprendidas

### Desarrollo con Streamlit

**Ventajas Identificadas:** - Desarrollo rápido de interfaces web sin conocimiento profundo de HTML/CSS/JavaScript - Integración natural con el ecosistema de Python y pandas - Facilidad para crear dashboards interactivos y visualizaciones - Deployment simplificado comparado con frameworks tradicionales

**Consideraciones Importantes:** - Limitaciones en personalización avanzada de UI - Manejo de estado requiere comprensión del modelo de Streamlit - Rendimiento puede verse afectado con grandes volúmenes de datos - Concurrencia limitada para aplicaciones de alta demanda

## Arquitectura de Datos

**Patrón Strategy Efectivo:** La implementación del patrón Strategy para manejo de múltiples fuentes de datos demostró ser altamente efectiva, proporcionando flexibilidad sin comprometer la simplicidad del código.

**Importancia del Fallback:** La capacidad de funcionar con Excel cuando SQL Server no está disponible ha demostrado ser crucial para la robustez del sistema.

## Gestión de Usuarios

**Simplicidad vs Seguridad:** Para el entorno de desarrollo y pruebas, se optó por un sistema de autenticación simplificado. Para producción, se recomienda implementar hashing de contraseñas y medidas de seguridad adicionales.

## Recomendaciones para Futuras Mejoras

### Mejoras de Seguridad

1. **Autenticación Avanzada:**
  2. Implementar hashing de contraseñas con salt
  3. Agregar autenticación de dos factores
  4. Implementar políticas de contraseñas seguras
  5. Sistema de bloqueo por intentos fallidos
6. **Autorización Granular:**
  7. Permisos específicos por funcionalidad
  8. Roles adicionales (supervisor, gerente, etc.)
  9. Auditoría completa de acciones de usuario

### Mejoras Funcionales

1. **Reportes Avanzados:**
  2. Generación de reportes en PDF
  3. Gráficos y visualizaciones avanzadas
  4. Exportación a múltiples formatos

5. Reportes programados automáticos

**6. Comunicación Mejorada:**

7. Sistema de chat en tiempo real

8. Notificaciones push

9. Integración con email

10. Recordatorios automáticos

**11. Gestión de Proyectos:**

12. Diagramas de Gantt

13. Dependencias entre actividades

14. Gestión de recursos

15. Seguimiento de presupuestos

## **Mejoras Técnicas**

**1. Rendimiento:**

2. Implementar cache de datos

3. Paginación para grandes datasets

4. Optimización de consultas

5. Lazy loading de componentes

**6. Escalabilidad:**

7. Migración a arquitectura de microservicios

8. Implementación de load balancing

9. Base de datos distribuida

10. API REST para integración externa

**11. Monitoreo:**

12. Logging estructurado

13. Métricas de aplicación

14. Alertas automáticas

15. Dashboard de monitoreo

## **Consideraciones de Implementación**

### **Para Organizaciones Pequeñas**

El sistema actual es ideal para organizaciones pequeñas a medianas que necesitan: - Solución rápida de implementar - Costos mínimos de infraestructura - Facilidad de mantenimiento - Flexibilidad en fuentes de datos

### **Para Organizaciones Grandes**

Para organizaciones más grandes, se recomienda: - Migración a base de datos empresarial - Implementación de medidas de seguridad avanzadas - Integración con sistemas existentes - Personalización de workflows específicos

## **Reflexión Final**

El desarrollo de este Sistema de Gestión de Empleados ha demostrado que es posible crear soluciones empresariales robustas y funcionales utilizando herramientas modernas de Python. La combinación de Streamlit para la interfaz, pandas para manipulación de datos, y una arquitectura bien diseñada ha resultado en un sistema que no solo cumple con los requisitos funcionales, sino que también proporciona una base sólida para futuras expansiones.

La experiencia de desarrollo ha reforzado la importancia de: - Diseño modular desde el inicio del proyecto - Consideración temprana de múltiples escenarios de uso - Implementación de patrones de diseño apropiados - Documentación completa para facilitar mantenimiento

Este proyecto sirve como ejemplo de cómo las herramientas modernas de desarrollo pueden democratizar la creación de aplicaciones empresariales, permitiendo que desarrolladores con diferentes niveles de experiencia puedan crear soluciones valiosas para organizaciones reales.

El sistema está listo para su implementación en entornos de producción con las consideraciones de seguridad apropiadas, y proporciona una base excelente para

futuras mejoras y expansiones según las necesidades específicas de cada organización.

---

**Fecha de Finalización:** 25 de Junio, 2025

**Versión del Documento:** 1.0

**Autor:** Manus AI

**Estado:** Completo y Listo para Producción